Ejercicios – Comunicación UDP

El siguiente código utiliza s*ockets* datagrama para intercambiar una única cadena de datos. La lógica del programa es lo más sencilla posible para subrayar la sintaxis básica de las comunicaciones entre procesos.

El emisor crea un paquete datagrama que contiene una dirección de destino, mientras que el paquete datagrama del receptor no incluye una dirección de destino.

```
import java.net.*;
import java.io.*;

public class Example1Sender {
  public static void main(String[] args) {
    if (args.length != 3)
      System.out.println ("This program requires three command line arguments");
    else {
      try {
        InetAddress receiverHost = InetAddress.getByName(args[0]);
        int receiverPort = Integer.parseInt(args[1]);
        String message = args[2];

        // instantiates a datagram socket for sending the data
        DatagramSocket  mySocket = new DatagramSocket();
        byte[ ] buffer = message.getBytes( );
        DatagramPacket datagram =
          new DatagramPacket(buffer, buffer.length,
                             receiverHost, receiverPort);
        mySocket.send(datagram);
        mySocket.close( );
      } // end try
      catch (Exception ex) {
        ex.printStackTrace( );
      }
    } // end else
  } // end main
} // end class
```

El *socket* del emisor se enlaza a un número de puerto no especificado, mientras que el *socket* del receptor se enlaza a un número de puerto específico, para que el emisor pueda escribir este número de puerto en su datagrama como destino.

```
import java.net.*;
import java.io.*;

public class Example1Receiver {
  public static void main(String[] args) {
    if (args.length != 1)
      System.out.println("This program requires a command line argument.");
    else {
      int port = Integer.parseInt(args[0]);
      final int MAX_LEN = 10;
      // This is the assumed maximum byte length of the datagram to be received.
      try {
        DatagramSocket  mySocket = new DatagramSocket(port);
        // instantiates a datagram socket for receiving the data
        byte[ ] buffer = new byte[MAX_LEN];
        DatagramPacket datagram = new DatagramPacket(buffer, MAX_LEN);
        mySocket.receive(datagram);
        String message = new String(buffer);
        System.out.println(message);
        mySocket.close( );
      } // end try
      catch (Exception ex) {
        ex.printStackTrace( );
      }
    } // end else
  } // end main
} // end class
```

1. Compila y ejecuta el código del ejemplo en una máquina usando "*localhost*" como nombre de máquina. Por ejemplo se puede introducir el comando:

   java Example1Sender localhost 12345 Hola

   Ejecuta los dos programas arrancando primero al receptor y después al emisor. El mensaje que se envíe no debería exceder la longitud máxima permitida que es de 10 caracteres.

   Describe el resultado de la ejecución.

2. Repite el ejercicio anterior utilizando máquinas diferentes a tu máquina local.

3. Vuelve a ejecutar las aplicaciones del apartado 1, esta vez ejecutando primero al emisor y luego al receptor.

   Describe y explica el resultado.

4. Repite el apartado 1, esta vez mandando un mensaje de longitud más grande que la máxima longitud permitida.

   Describe y explica la salida producida.

5. Añade código al proceso receptor de manera que el plazo máximo de bloqueo del receive sea de cinco segundos. Lanza el proceso receptor pero no el

proceso emisor.

¿Cuál es el resultado? Descríbelo y explícalo.

6. Modifica el código original de manera que el receptor ejecute indefinidamente un bucle que reciba y muestre los datos recibidos. Compílalo y ejecútalo de la siguiente forma:

    - Lanza al receptor
    - Ejecuta el emisor enviando un mensaje ``mensaje 1"
    - En otra ventana, lanzar otra instancia del emisor, mandando un mensaje ``mensaje 2".

    Describe y explica el resultado.

7. Modifica el código original de manera que el emisor utilice el mismo *socket* para enviar el mismo mensaje a dos receptores diferentes. Primero lanza los dos receptores y después al emisor. ¿Cada receptor recibe el mensaje? Describe y explica el resultado.

8. Modifica el código original de manera que el emisor utilice dos *socket* distintos para enviar el mismo mensaje a dos receptores diferentes. Primero lance los dos receptores y después al emisor.

    ¿Cada receptor recibe el mensaje?

    Describe y explica el resultado.

9. Modifica el código del último paso de modo que el emisor envíe de forma permanente, suspendiéndose durante 3 segundos entre cada envío.

    Modifica el receptor de manera que ejecute un bucle que repetidamente reciba datos y luego los muestre.

    Compila y ejecuta los programas durante unos cuantos segundos antes de terminarlos con ``Ctrl-C".

    Describe y explica el resultado

10. Modifica el código original de modo que el emisor también reciba un mensaje del receptor. Utilizar sólo un *socket* en cada proceso. Entrega este código.